IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applic. Serial No.:	10/696,393	
Filing Date:	October 29, 2003	
Applicant:	Colt R. Correa	**Appeal Brief**
Examiner:	Zheng Wei	
Group Art Unit:	2192	
Title:	METHOD FOR ECU CALIBRATION AND DIGANOSTIC DEVELOPMENT	
Attorney Docket No.	2485-000001/CPA	

BRIEF ON BEHALF OF APPELLANTS AND PETITION FOR EXTENSION OF TIME

This is a brief in support of an appeal from the action of the Examiner dated May 14, 2008, rejecting Claims 1, 5-8, 10-12 and 14-19 of the present application. Copies of the appealed claims are attached as an appendix.

Applicant hereby petitions for an extension of time in which to submit an appeal brief and includes a fee as set forth in 37 C.F.R. § 1.17 with this brief for such extension of time.

TABLE OF CONTENTS

I. Real Party In Interest

The real party in interest in the present application is Accurate Technologies, Inc. who is the current assignee of the application.

II. Related Appeals and Interferences

There are no known related appeals or interferences which will directly affect, be directly affected by, or otherwise have a bearing on the Board's decision in the pending appeal.

III. Status of the Claims

Claims 1, 5-8, 10-12 and 14-19 are pending in the present application. Claims 1, 5-8, 10-12 and 14-19 stand rejected as indicated in the Office Action mailed on May 14, 2008. Claims 2-4, 9 and 13 are cancelled. Claims 1, 5-8, 10-12 and 14-19 are the subject of this appeal.

IV. Status of Amendments

Claims 1, 5-8, 10-12 and 14-19 stand as presented in applicant's response dated January 30, 2008.

V. Summary of the Claimed Subject Matter

Applicant's invention relates to a method and system that enables calibration tools to overwrite user selected random access memory variables without any modifications or access to the underlying ECU source code. A concise explanation of the subject matter defined in each independent claim and each dependent claim argued separately is provided below. The explanation refers to the specification as filed by page, paragraph, and paragraph line number and to the drawings by reference numbers. The citations to the application are for exemplary purposes only as the invention includes numerous embodiments.

CLAIM 1

Independent Claim 1 recites a method for controlling a value of a RAM variable inside an executable program (e.g. [Fig. 1; page 5 at ¶[0015], lines 1-4), comprising:

presenting a software program in executable form (e.g. Fig. 1, step 30; page 5 at ¶[0016], lines 1-3) and having a plurality of machine instructions of a finite quantity of fixed lengths (e.g. page 5 at ¶[0016], lines 1-3);

identifying at least one machine instruction that accesses a variable defined in random access memory associated with the software program (e.g. page 6 at ¶[0017], line 3; page 6 at ¶[0018], lines 1-11);

replacing the identified machine instruction in the executable form of the software program (e.g. Fig 1, step 34, page 6 at ¶[0017], lines 2-3) with a branch instruction that references an address outside an address space of the software program (e.g. page 7 at ¶[0020], lines 2-10);

defining a set of relocated instructions at the address referenced by the branch instruction (e.g. pages 8-9 at ¶[0024], lines 3-5), wherein the set of relocated instructions function to change a value of the variable (e.g. page 9, ¶[0026]-¶[0028]); and

executing the executable form of the software program having the branch instruction (e.g. page 15 at ¶[0039], lines 10-11).

CLAIM 10

Independent Claim 10 recites a computer-implemented calibration system for modifying RAM variables of a software program embedded in a microprocessor (e.g. Fig. 4; page 11 at ¶[0032], lines 1-7), comprising:

an instruction locator embodied as computer executable instructions on a computer readable medium and operating on a different processor than the microprocessor (e.g. Fig. 4, object 76), the instruction locator adapted to receive an address for RAM variable within an software program (e.g. page 11 at ¶[0032), lines 3-7] and operable to identify at least one machine instruction in an executable form of the software program that accesses the RAM variable (e.g. page 11 at ¶[0032], lines7-9); and

an instruction replacement component embodied as computer executable instructions on a computer readable medium and operating on the different processor than the microprocessor and in data communication with the instruction locator (e.g. Fig. 4 object 78), the instruction replacement component adapted to receive a branch instruction for the at least one machine instruction (e.g. page 12 at ¶[0034], lines 2-4) and operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction (e.g. page 12 at ¶[0034, lines 2-5).

CLAIM 14

Dependent Claim 14 depends on Claim 10 and recites the computer-implemented system of Claim 10 wherein the instruction replacement component is operable to generate a set of relocation instructions (e.g. pages 12-13 at ¶[0034], lines 5-8), such that the branch instruction passes processing control to the set of relocation instructions (e.g. page 7 at ¶[0019] lines 4-7).

CLAIM 16

Independent Claim 16 recites a method for controlling the value of a RAM variable inside an executable program (e.g. Fig. 1; page 5 at ¶[0015], lines 1-4), comprising:

presenting a software program in executable form (e.g. Fig. 1, step 30; page 5 at ¶[0016], lines 1-3) and having a plurality of machine instructions of a finite quantity of fixed lengths (e.g. page 5 at ¶[0016], lines 1-3);

identifying at least two machine instructions that accesses a variable defined in random access memory associated with the software program (e.g. page 6 at ¶[0017], line 3; page 6 at ¶[0018], lines 1-11);

replacing each of the identified machine instruction in the executable form of the software program with a branch instruction (e.g. Fig 1, step 34, page 6 at ¶[0017], lines 2-3), where each branch instruction references a different address outside an address space of the software program (e.g. page 7 at ¶[0020], lines 2-10);

defining a set of relocated instructions at the address referenced by the branch instruction (e.g. pages 8-9 at ¶[0024], lines 3-5), wherein the set of relocated instructions

accesses the variable in random access memory and performs an operation to change a value of the variable (e.g. page 9, ¶[0026]-¶[0028]); and

executing the executable form of the software program having the branch instruction (e.g. page 15 at ¶[0039], lines 10-11).

VI. Grounds of Rejection to be Reviewed on Appeal

1. Whether the Karp et. al. reference renders Claims 1, 5-8, 10-12 and 14-19 obvious under 35 U.S.C. § 103(a).

VII. Arguments

1. THE KARP ET AL. REFERENCE FAILS TO RENDER THE INVENTION OF CLAIMS 1 AND 5-8 OBVIOUS UNDER 35 U.S.C. § 103(a).

Claims 1 and 5-8 stand rejected under 35 U.S.C. §103(a) in light of the U.S. 2003/0061598, Karp et al. reference. (hereinafter "Karp"). Applicant respectfully submits that Karp fails to render Claim 1, and the claims depending therefrom, obvious. In particular, (i) Karp discloses a method that does not change the functionality of the source code; (ii) Karp does not teach replacing identified machine instructions with a branch instruction that references an address outside an address space of the software program; (iii) Karp does not teach defining a set of relocated instructions referenced by the branch instructions, wherein the set of relocated instructions function to change a value of the variable; and (iv) Karp's disclosure of a "break instruction and a branch instruction together" instruction does not render a "branch" instruction obvious.

First, Karp does not disclose and is not directed to a method that changes the functionality of source code. Rather, Karp teaches a system with "mechanisms for providing hint instructions to a processor without altering object code instruction sequences." (Karp, page 1 at ¶ [0009], lines 1-3) (emphasis added). Karp discloses an "object code adapter which provides hint instructions to the processor using a mechanism for handling break instructions." (Karp, page 1 at ¶[0018], lines 3-6). The object code adapter "adapts the object code by providing hint instructions in place of selected instructions in the object code." (page 1 at ¶[0021], lines 1-3). Karp provides an example, stating "the object code adapter replaces the instruction I_3 with a break instruction B_1." (Karp, page 1 at ¶[0021], lines 3-5). Karp, however, requires the execution of instruction I_3, as "the hint code is code to be executed by the processor when

the break instruction B_1 is executed. The hint code includes a hint instruction H_1 and the instruction I_3 that was replaced by the break instruction..." (Karp, page 1 at ¶[0022] at 1-4) (emphasis added). Thus, Karp does not contemplate replacing an instruction I_n with another instruction, but rather contemplates inserting a hint instruction in an instruction stream or pipeline.

Conversely, Claim 1 recites, in part, "identifying at least one machine instruction that accesses a variable defined in random access memory associated with the software program" and "replacing the identified machine instruction in the executable form of the software program with a branch instruction that references an address outside an address space of the software program." (emphasis added). Thus, Claim 1 requires the replacement of one instruction with a second instruction. The second instruction, the branch instruction, changes the control flow of the executable code by referencing an address outside the address space of the software program. (e.g. page 7 at ¶[0019], lines 4-7). Claim 1 further recites "defining a set of relocated instructions at the address referenced by the branch instruction." Therefore, when the branch instruction is executed, the relocated instructions are also executed as the branch instruction changes the control flow of the program so as to include the instructions contained in the referenced address in lieu of the replaced instruction. This point of distinction highlights that Karp contemplates an unrelated objective, namely providing hint instructions to a processor, rather than changing the functionality of software program already in executable code.

Second, Karp does not teach replacing identified machine instructions with a branch instruction that references an address outside an address space of the software

program. As discussed above, Karp essentially teaches inserting hint code in the object code. Karp teaches "inserting break instructions in place of selected instructions in the object code." (Karp, page 1 at ¶[[0021] lines 2-3). "The hint code is to be executed by the processor when the break instruction is executed. The hint instruction includes a hint instruction and the instruction that was replaced by the break instruction." (Karp, page 1 at ¶[[0022], lines 1-2). Figure 3 of Karp details the handling of a break instruction. "At step 120, the processor obtains a hint instruction from the hint register and inserts the hint instruction into the stream to be executed." (Karp, page 3 at [0037], lines 2-4). The next step recites "the processor obtains the replaced instruction I_X from the hint register and inserts it into instruction stream to be executed." (Karp, page 3 at [0037], lines 5-7). Thus, Karp expressly contemplates the execution of the "replaced" instruction.

Conversely, the method recited in Claim 1 actually replaces the replaced instruction with relocated instructions. (Page 6, at ¶[[0017], lines 3-4). Claim 1 recites "identifying at least one machine instruction that accesses a variable defined in random access memory associated with the software program." This is the instruction that is replaced by a branch instruction, as each replacement instruction is preferably defined as a branch instruction that references an address outside the memory space for the target software program. (page 7, at ¶[[0019], lines1-4). Accordingly, Claim 1 further recites "replacing the identified machine instruction in the executable form of the software program with a branch instruction that references an address outside an address space of the software program."

Thus, Karp and Claim 1 are entirely different, as a replaced instruction, according to Karp, will be executed and is therefore not actually replaced; while a replaced

instruction, according to Claim 1, will not be executed and is actually replaced by a branch instruction. Furthermore, non-execution of a replaced instruction is not an obvious variation of execution of a replaced instruction, as the execution of a single instruction may alter the control flow or functionality of an application. Exclusion of a single instruction may further result in run-time errors. Thus, one having skill in the art would not contemplate using Karp's insertion of a hint instruction instead of Claim 1's replacement of an instruction or vice-versa.

Third, Karp does not teach defining a set of relocated instructions referenced by the branch instructions, wherein the set of relocated instructions function to change a value of the variable. Karp contemplates the use of a hint instruction. A hint instruction is not the functional equivalent of a relocated instruction. "One example of a hint instruction is a pre-fetch instruction that includes a pre-fetch address. The processor executes a pre-fetch instruction by fetching a set of data from memory using the pre-fetch address and writing the data into a cache associated with the processor." (Karp, page 2 at ¶[0023], lines 1-5). In this example, a hint instruction is increasing the efficiency of the processor by inserting a pre-fetch instruction, which fetches the originally contemplated data. In another example, "a hint instruction is a branch prediction that specifies a likely result of a branch instruction in the sequence of I_1 through I_N." (Karp, page 2 at ¶[0024], lines 1-3). The two above-referenced examples, and variations thereof, are the only two examples of hint instructions provided. Thus, Karp does not teach a set of relocated instructions that function to change a value of a RAM variable inside an executable code.

Furthermore, Karp, in disclosing the background of the invention, states "it is common for processor manufacturers to provide a family of processors that conform to a

given macro-architecture. Processors in a family usually vary according to micro-architecture features such as on-chip caches, out-order processing capabilities, [or] branch prediction capabilities." (Karp, page 1 at ¶[0006], lines 1-5) Karp continues to define the problem presented: "Unfortunately, object code which is compiled based on the micro-architecture features on one member of a processor family may suffer in performance when executed on another member of the family." (Karp, page 1 at ¶[0007], lines 1-4). Thus, Karp's objective is increasing a processor's efficiency when executing object code by means of inserting hint instructions to change a value of the variable. Moreover, Claim 1's relocated instructions are not obvious variations of Karp's hint instructions. As discussed, the two types of instructions or codes are entirely different. A finding to the contrary would essentially render all assembly code instructions as obvious variations of one another. Accordingly, Karp teaching the use of hint instructions does not render obvious the use of relocated instructions that function to change a value of the variable.

Finally, Karp's recital of a break instruction and branch together does not provide the same feature/functionally of the branch instruction recited in Claim 1. The Examiner asserts that "the implementation of Karp using the break instruction and branch instruction together provides the same feature/functionality to use redirect/branch instruction to jump to a different memory address. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use Karp's implementation to reference a different memory address." It is respectfully submitted that Karp discloses that "the hint code may include a branch or return instruction to resume execution of the object code **depending on the implementation of**

the break mechanism in the processor." (Karp, pages 1-2 at ¶[0022], lines 7-10) (emphasis added). Thus, the use of "a break instruction and branch instruction together" is not used in Karp to reference a memory address referencing a set of relocation instructions. Rather, Karp contemplates the use of the branch instruction in the event a return instruction is not supported by the processor. The branch instruction is used to return control to the object code, rather than to reference the address of relocation code.

In contrast, Claim 1 replaces an instruction with a branch instruction so that control is relinquished to the relocation code. Thus, the use of the branch instruction in Claim 1 is the functional equivalent of inserting the relocation code in lieu of the replaced instruction. Moreover, an instruction that causes an interrupt or an exception, i.e. a break instruction, are "generally not preferred because these types of instructions may interfere with the normal operation of the microprocessor." (page 7 at ¶[0020], lines 5-8). So while a break instruction may be used to achieve the Karp's above-stated objective, the use of such an instruction to is generally not desirable or preferable to control the value of a RAM variable.

Thus, the Examiner's assertion that the use of the break instruction and branch together provides the same functionality is incorrect. Accordingly, one having skilled in the art would not use Karp's implementation—that is, a break instruction for the purposes of executing a hint instruction and a replaced instruction and using a branch instruction to return to the next instruction in the instruction stream—to achieve replacing the identified machine instruction in the executable form of the software program with a branch instruction that references an address outside an address space of the software program.

In view of the foregoing arguments, Applicant respectfully submits that Claim 1 and the claims depending therefrom are not rendered obvious by Karp. Applicant respectfully requests that the Board reverse the Examiner's rejection of the claims.

2. THE KARP ET AL. REFERENCE FAILS TO RENDER THE INVENTION OF CLAIMS 10-12 AND 14-15 OBVIOUS UNDER 35 U.S.C. § 103(a).

The Examiner asserts that "Claims 10-12 and 14-15 are system version for performing the claimed method as in Claims 1 and 5-8 addressed above, wherein all claimed limitation functions have been addressed and/or set forth above and certainly a computer system would need to run and/or practice such function steps disclosed by reference above. Thus, they also would have been obvious." In addition to the arguments presented with respect to Claim 1, Applicant respectfully submits that the Examiner's characterization of Independent Claim 10, and the claims depending therefrom, is incorrect. Specifically, Karp does not teach an instruction replacement component that is "adapted to receive a branch instruction for the at least one machine instruction and operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction."

Claim 10 recites, in part, an "instruction replacement component adapted to receive a branch instruction for the at least one machine instruction and operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction." As discussed above, Karp does not actually teach the replacement of an instruction. Rather, Karp teaches "inserting break instructions in place of selected instructions in the object code." (Karp, page 1 at ¶[0021] lines 2-3). "The hint code is to be executed by the processor when the break instruction is executed. The hint instruction includes a hint instruction and the instruction that was replaced by

the break instruction." (Karp, page 1 at ¶[0022], lines 1-2). Figure 3 of Karp details the handling of a break instruction. "At step 120, the processor obtains a hint instruction from the hint register and inserts the hint instruction into the stream to be executed." (Karp, page 3 at [0037], lines 2-4). The next step recites "the processor obtains the replaced instruction I_X from the hint register and inserts it into instruction stream to be executed." (Karp, page 3 at [0037], lines 5-7). Thus, Karp expressly contemplates the execution of the "replaced" instruction.

Conversely, Claim 10 teaches an instruction replacement component "operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction." As discussed above, in order to "manipulate a given RAM variable, one or more machine instructions having access to the variable are replaced with replacement instructions." (Page 6 at [0017], line 1-3). "The relocated instructions are preferably machine instructions that access or modify variables that correspond to the RAM variables that are desired to be controlled." (page 6 at [0018] lines 2-3). Due to the nature of the task being completed, i.e. manipulating a RAM variable, it is undesirable to execute the replacement instruction as well as the instruction that is to be replaced. Accordingly, Karp cannot be read so as to teach an instruction replacement component "operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction." (Claim 10). As discussed above, actual replacement of an instruction and the insertion of a hint instruction are not obvious variations of one another. Using one method interchangeably with the other method may reduce efficiency, cause run time errors, or undesirably alter the functionality of the code. Therefore, Claim 10 and the claims depending therefrom

are not rendered obvious by Karp. Applicant respectfully requests that the Board reverse the Examiner's rejection of the claims.

3. THE KARP ET AL. REFERENCE FAILS TO RENDER THE INVENTION OF CLAIMS 14-15 OBVIOUS UNDER 35 U.S.C. § 103(a).

Applicant respectfully submits that Karp does not render Dependent Claim 14, and the claim depending therefrom, obvious. In particular, Karp does not disclose an instruction replacement component "operable to generate a set of relocation instructions, such that the branch instruction passes processing control to the set of relocation instructions.

Claim 14 depends on Claim 10 and further recites an instruction replacement component "operable to generate a set of relocation instructions, such that the branch instruction passes processing control to the set of relocation instructions." The object code adapter taught in Karp does not generate relocation instructions. Rather, the object code adapter provides "hint instructions" to the processor using a mechanism for handling break instructions which is built in to the processor and a hint register contained in the processor. (page 2 at ¶[0030], lines 4-7). This implies that the hint instructions are predetermined and stored in the processor before the object code adapter commences operation. In any event, Karp does not disclose an embodiment of the object code adapter that is operable to generate hint instructions.

Claim 14, however, teaches an instruction replacement component that generates a set of relocation instructions. In one example, the relocation code may be generated such that the relocation code "reads the desired value of the RAM variable from a location which may be modified by the calibration system. The relocation code then stores the desired value into the original RAM variable." (page 9 at ¶[0026], lines 2-5).

see also fig 3A) The generated relocation code "branches processing...to the machine instruction following the relocated instruction in the target software program." (page 9 at ¶[0027], lines 1-3). Alternatively, the relocation code may be generated to support an indirect addressing method for storing or loading variable values. (pages 9-10 at ¶[0028], lines 1-3). In this embodiment, the replacement code should also determine the register that points to the variable and then search for all store instructions using that register. (page 10 at ¶[0028], lines 5-8). In this indirect addressing mode, it is necessary for the relocation code to determine that the value of the register is in fact pointing to the given variable. (page 10 at ¶[0029], lines 2-4). Accordingly, the instruction replacement component claimed in Claim 14 and supported in the detailed description is not taught by Karp. Furthermore, the instruction replacement component is not rendered obvious by Karp's object code adapter as Karp does not mention, contemplate or otherwise suggest that a hint instruction generating functionality is possible or contemplated. Thus, Claim 14 and the claims depending therefrom are not rendered obvious by Karp. Applicant respectfully requests that the Board reverse the Examiner's non-final rejection of the claims.

4. THE KARP ET AL. REFERENCE FAILS TO RENDER THE INVENTION OF CLAIMS 16-19 OBVIOUS UNDER 35 U.S.C. § 103(a).

Claim 16, and the claims depending therefrom, stands rejected under 35 U.S.C. §103(a) in light of the Karp et al. reference. Applicant respectfully submits that Karp reference fails to render Claim 16 obvious. Namely, Applicant Claim 16 recites subject matter that is very similar to Independent Claim 1. The primary distinctions between Claim 1 and Claim 16 are that Claim 16 has a step of "identifying at least two machine instructions that accesses a variable defined in random access memory associated with

the software program," instead of "identifying at least one machine instruction that accesses a variable defined in random access memory associated with the software program;" and Claim 16 recites a step of "defining a set of relocated instructions at each address referenced by the branch instructions, wherein each set of relocated instructions accesses the variable in random access memory and performs an operation to change a value of the variable in a different manner," while Claim 1 recites "defining a set of relocated instructions at the address referenced by the branch instruction, wherein the set of relocated instructions function to change a value of the variable." Thus, all of the arguments provided in section 1 are applicable to Claim 16. Namely, (i) Karp discloses a method that does not change the functionality of the source code; (ii) Karp does not teach replacing identified machine instructions with a branch instruction that references an address outside an address space of the software program; (iii) Karp does not teach defining a set of relocated instructions referenced by the branch instructions, wherein the set of relocated instructions accesses the variable in random access memory and performs an operation to change a value of the variable; (iii) Karp's disclosure of a "break instruction and a branch instruction together" instruction does not render a "branch" instruction obvious. Accordingly, for the reasons presented above, Independent Claim 16 and the claims depending therefrom are patentable in view of Karp.

For the foregoing reasons, the appealed claims are patentable over the art relied upon by the Examiner. Accordingly, Applicant's representative respectfully requests that this Board reverse the non-final rejection of Claims 1, 5-8, 10-12 and 14-19.

Respectfully submitted,

Dated: November 13, 2008

By: /Bryant E. Wade/
Timothy D. MacIntyre
Reg. No. 42,824
Bryant E. Wade
Reg. No. 40,344

HARNESS, DICKEY & PIERCE, P.L.C.
P.O. Box 828
Bloomfield Hills, Michigan 48303
(248) 641-1600

TDM/BEW/TET

VIII. Claims Appendix

1. A method for controlling a value of a RAM variable inside an executable program, comprising:

presenting a software program in executable form and having a plurality of machine instructions of a finite quantity of fixed lengths;

identifying at least one machine instruction that accesses a variable defined in random access memory associated with the software program;

replacing the identified machine instruction in the executable form of the software program with a branch instruction that references an address outside an address space of the software program;

defining a set of relocated instructions at the address referenced by the branch instruction, wherein the set of relocated instructions function to change a value of the variable; and

executing the executable form of the software program having the branch instruction.

2. cancel

3. cancel

4. cancel

5. The method of Claim 1 wherein the step of identifying at least one machine instruction further comprises determining location information for the at least one machine instruction within the software program.

6. The method of Claim 5 wherein the step of determining location information further comprises identifying an address for the at least one machine instruction using the image of the executable containing the machine instructions that comprise the executable.

7. The method of Claim 6 wherein the step of replacing the at least one machine instruction further comprises inserting the replacement instruction into a program memory image of the software program at said address.

8. The method of Claim 1 wherein said branch instruction references a set of relocation instruction residing at an unused address space of the software program.

9. cancel

10. A computer-implemented calibration system for modifying RAM variables of a software program embedded in a microprocessor, comprising:

an instruction locator embodied as computer executable instructions on a computer readable medium and operating on a different processor than the microprocessor, the instruction locator adapted to receive an address for RAM variable

within an software program and operable to identify at least one machine instruction in an executable form of the software program that accesses the RAM variable; and

an instruction replacement component embodied as computer executable instructions on a computer readable medium and operating on the different processor than the microprocessor and in data communication with the instruction locator, the instruction replacement component adapted to receive a branch instruction for the at least one machine instruction and operable to replace the at least one machine instruction in the executable form of the software program with the branch instruction.

11. The computer-implemented system of Claim 10 wherein the instruction locator is operable to identify an address for the specified machine instruction using the image of the executable containing the machine instructions that comprise the executable.

12. The computer-implemented system of Claim 11 wherein the instruction replacement component is operable to insert the replacement instruction into a program memory image of the software program at said address.

13. cancel

14. The computer-implemented system of Claim 10 wherein the instruction replacement component is operable to generate a set of relocation instructions, such that the branch instruction passes processing control to the set of relocation instructions.

15. The computer-implemented system of Claim 14 wherein the instruction replacement component is further operable to insert the set of relocation instructions in a memory space of the microprocessor that resides at an unused address space of the software program.

16. A method for controlling the value of a RAM variable inside an executable program, comprising:

presenting a software program in executable form and having a plurality of machine instructions of a finite quantity of fixed lengths;

identifying at least two machine instructions that accesses a variable defined in random access memory associated with the software program;

replacing each of the identified machine instruction in the executable form of the software program with a branch instruction, where each branch instruction references a different address outside an address space of the software program;

defining a set of relocated instructions at each address referenced by the branch instructions, wherein each set of relocated instructions accesses the variable in random access memory and performs an operation to change a value of the variable in a different manner; and

executing the executable form of the software program having the branch instruction.

17. The method of Claim 16 wherein the step of identifying at least one machine instruction further comprises determining location information for the at least one machine instruction within the software program.

18. The method of Claim 16 wherein the step of determining location information further comprises identifying an address for the at least one machine instruction using the image of the executable containing the machine instructions that comprise the executable.

19. The method of Claim 18 wherein the step of replacing the at least one machine instruction further comprises inserting the replacement branch instruction into a program memory image of the software program at said address.

IX. Evidence Presented

None.

X. Related Proceedings Appendix

None.